Multi-Agent Programming

Brian Logan

School of Computer Science
University of Nottingham

Midlands Graduate School
8th – 12th April 2013

Course Overview

Lecture 1: *Programming agents*
BDI model; PRS and other BDI languages

Lecture 2: *Programming multi-agent systems*
Coordination in MAS; agent communication languages & protocols; programming with obligations and prohibitions

Lecture 3: *Logics for MAS*
LTL, CTL; Rao and Georgeff's BDI logics; Coalition Logic, ATL

Lecture 4: *Verification of MAS*
A tractable APL and BDI logic: SimpleAPL and PDL-APL

Lecture 1: Programming agents

Outline of this lecture

- what is an agent
- the belief-desire-intention (BDI) model of agency
- why agents need special programming languages
- elements of a BDI agent programming language

What is an agent?

Multi-agent programming

Multi-agent systems are a promising approach to constructing complex software systems that are:

- **Open:** agents dynamically enter and exit the system
- **Autonomous:** agents pursue their own objectives
- **Encapsulated:** internal state and operation of agents is not visible to other agents (or the MAS)
- **Heterogeneous:** agents can have different capabilities and be implemented in different ways (e.g., different agent programming languages)

Applications of (multi-)agent systems

- autonomous vehicles, e.g., Google car, UAVs etc.
- personal assistants, ambient intelligence
- workflow & logistics, e.g., warehouse management, maintenance scheduling
- real-time control & decision support (process monitoring, disaster management)
- healthcare, e.g., care coordination, patient monitoring
- power engineering, e.g., smart grids, VPPs
- information integration, e.g., sensor networks
- virtual environments, e.g., games, training, modelling and simulation

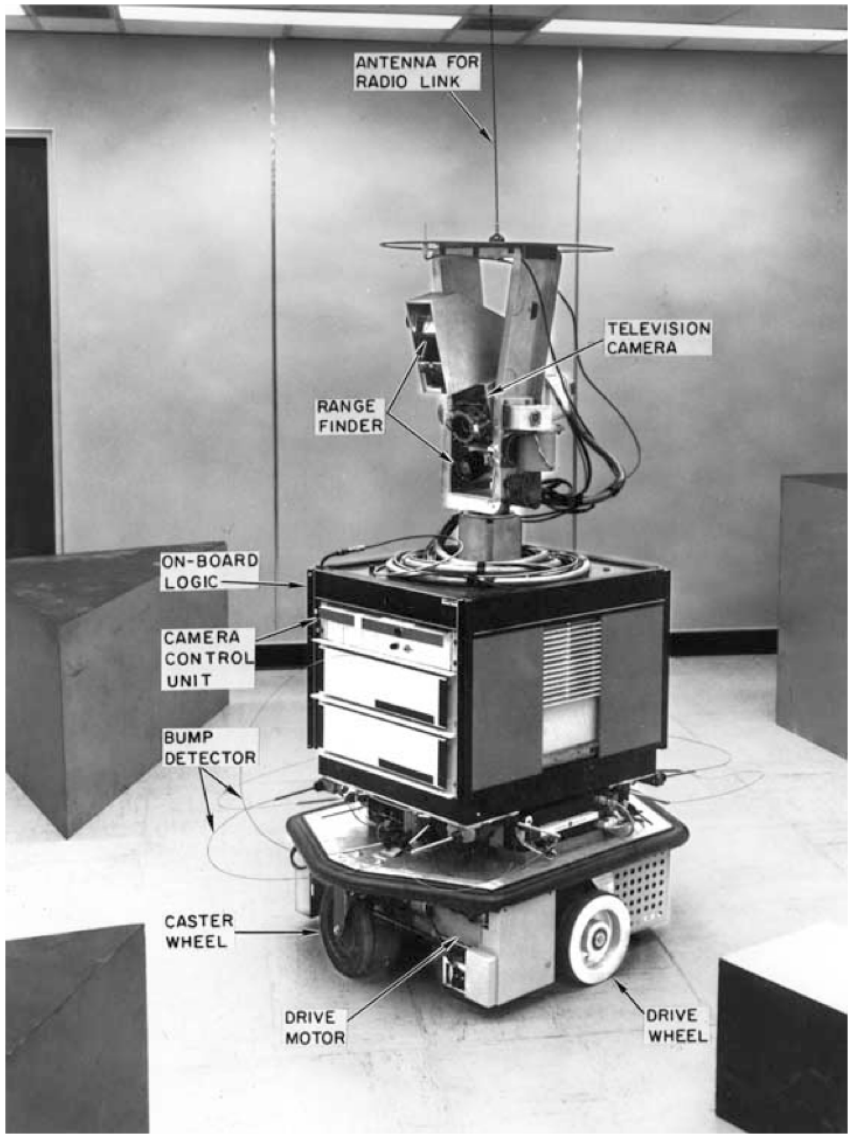
What is an agent?

Multi-agent systems are composed of agents

- many definitions of 'agent' in the multi-agent systems literature — key ideas include:
- **autonomy:** an agent operates without the direct intervention of humans or other agents
- **situatedness:** an agent interacts with its environment (which may contain other agents)
- **reactivity:** an agent responds in a timely fashion to changes in its environment
- **proactivity:** an agent exhibits goal-directed behaviour

Shakey the robot (1966–1972)

- Shakey was the first mobile robot to reason about its actions
- multiple sensors (TV camera, a triangulating range finder, and bump sensors)
- connected to DEC PDP-10 and PDP-15 computers via radio and video links
- programs for perception, world-modeling, and acting (simple motion, turning, and route planning)



What is different about agents?

- many complex computational systems exhibit some degree of autonomy, situatedness, reactivity, and proactivity — how to decide what's 'really' an agent
- an alternative way to think of agents is not in terms of their behaviour but in terms of:
- *how they are characterised*
- *how they are programmed*

Agents as intentional systems

For the purposes of this course, an agent can be defined as:

Definition (Agent)

- a computational system whose behaviour can be usefully characterised in terms of propositional attitudes such as beliefs and goals; and
- which is programmed in an agent programming language that makes explicit use of propositional attitudes

Digression: ascribing propositional attitudes

- not all agents represent beliefs and goals explicitly, even though they act in a goal-directed manner
- e.g., the behaviour of an agent may be controlled by a collection of decision rules which simply respond to the agent's current environment
- however it can still be useful to view the agent as an *intentional system* — that is we ascribe to it the beliefs and goals it *ought* to have, given what we know of its environment, sensors and (putative) desires (Dennett 1987, 1996)
- e.g., an agent which avoids obstacles can be said to have a goal of "avoiding collisions" even though this goal is not explicitly represented in the agent

The *intentional stance*

- Dennett (1987) calls this approach *"adopting the intentional stance"*
- allows us to ascribe propositional attitudes to agents which do not explicitly represent beliefs and goals
- intentional stance is more likely to yield useful insights than a description couched in terms of the low-level details of the the agent's implementation
- we will focus on agent programming languages in which the agent's state contains explicit representations of propositional attitudes

BDI model of agency

The *Belief Desire Intention* model of agency

- the belief–desire–intention model (Bratman 1987) stresses the importance of committing to plans to limit the amount of time an agent spends deliberating
- desires (states the agent wants to bring about) and intentions (states the agent has committed to bring about) are both pro-attitudes, but intentions constrain future behaviour
- commitment results in the temporal persistence of plans and future plans being made on the assumption that current plans will persist
- e.g., if I plan to attend MGS 2013, I will take this into account when scheduling a research project meeting — I don't (usually) reconsider all my plans when a new task or opportunity presents itself

What is an agent programming language?

- most work in *agent programming* assumes that agents are not only conceptualised in terms of beliefs, desires and intentions, but are also implemented in terms of beliefs, desires and intentions
- *agent programming languages* are designed to facilitate the implementation of BDI agents:
  - *programming constructs* corresponding to beliefs, desires and intentions
  - agent *architecture* or *interpreter* which enforces relationships between beliefs, desires and intentions and which causes the agent to choose actions to achieve its goals based on its beliefs

Why do we need agent programming languages?

- it is possible to implement agents in any programming language (C, Lisp, Prolog, Java, etc.)
- there are also a wide range of *agent toolkits* such as JADE, which provide basic facilities for representing agents
- however general purpose programming languages (and the agent toolkits based on them) don't implement the semantics of the BDI model of agency
- in these languages & toolkits, the agent programmer is responsible for implementing beliefs, desires and intentions more or less from scratch

The BDI agent programming model

- in contrast, agent programming languages (APLs) are defined in terms of beliefs, desires and intentions
- however most agent programming languages depart to some degree from the BDI model of agency defined by Bratman et al.:
  - expressiveness of beliefs, goals and plans
  - relationship between desires and goals
  - relationship between goals and beliefs
  - commitment to and processing of intentions

Procedural Reasoning System

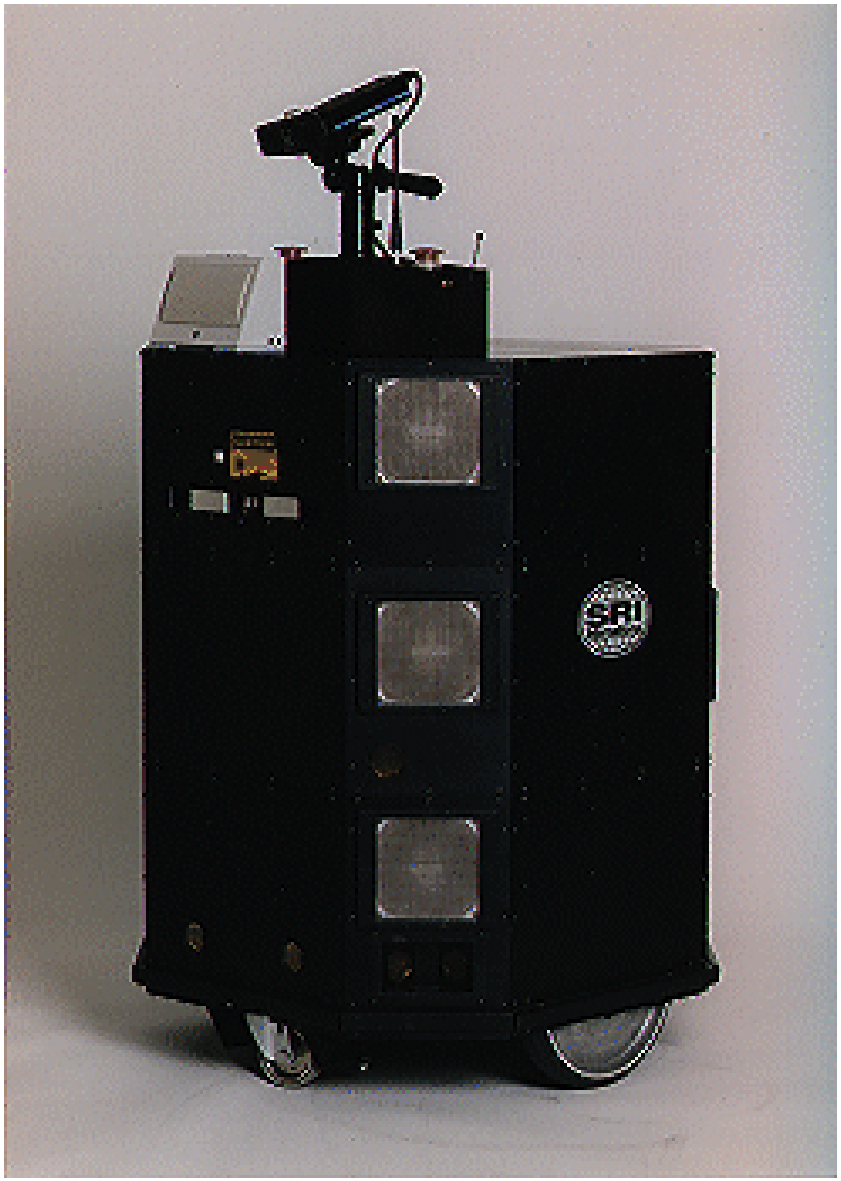
Procedural Reasoning System (PRS)

- PRS is arguably the first implementation of a belief-desire-intention architecture
- *"it should be possible to ascribe beliefs, goals and intentions to the system and to interact with it in terms of these psychological attitudes"* (Georgeff & Ingrand 1988)
- example applications include space shuttle fault diagnosis, controlling a mobile robot, air traffic control, business process control etc.
- very influential — many derivatives (PRS-CL, PRS-Lite, dMARS, AgentSpeak(L), SPARK, Jason)
- illustrates many of the key ideas in agent programming language design

PRS design objectives

- the system architecture should be both goal-directed and reactive
  - while working to attain specific goals the system should be able to react appropriately to new situations in real time
  - it should be able to completely alter focus and goal priorities as situations change
- in addition it should be able to reflect on its own reasoning processes
  - it should be able to choose when to change goals, when to plan and when to act and how to effectively use its deliberation capabilities

Flakey the robot (1992–1993)



PRS programs

- a PRS program consists of beliefs, (top-level) goals, plans and intentions
- *beliefs* represent things which the agent believes are currently true about the environment or itself
- *goals* represent desired behaviours of the system expressed as conditions over an interval of time, i.e., over a sequence of world states
- *plans* (Knowledge Areas) are predefined sequences of actions and tests which achieve a particular goal or react to a particular situation
- *intentions* are plans which have been chosen for execution

PRS beliefs

- beliefs can refer to both static and dynamic information
- static information describes fixed properties of the application domain, e.g., physical laws
- dynamic information changes with time, e.g., current percepts and conclusions derived by PRS
- beliefs can also describe the internal state of PRS — meta-level beliefs are used to describe the current goals and intentions of the agent, and the plans being considered for execution etc.
- meta-level beliefs play an important role in specifying the deliberation (control) strategy of a PRS agent

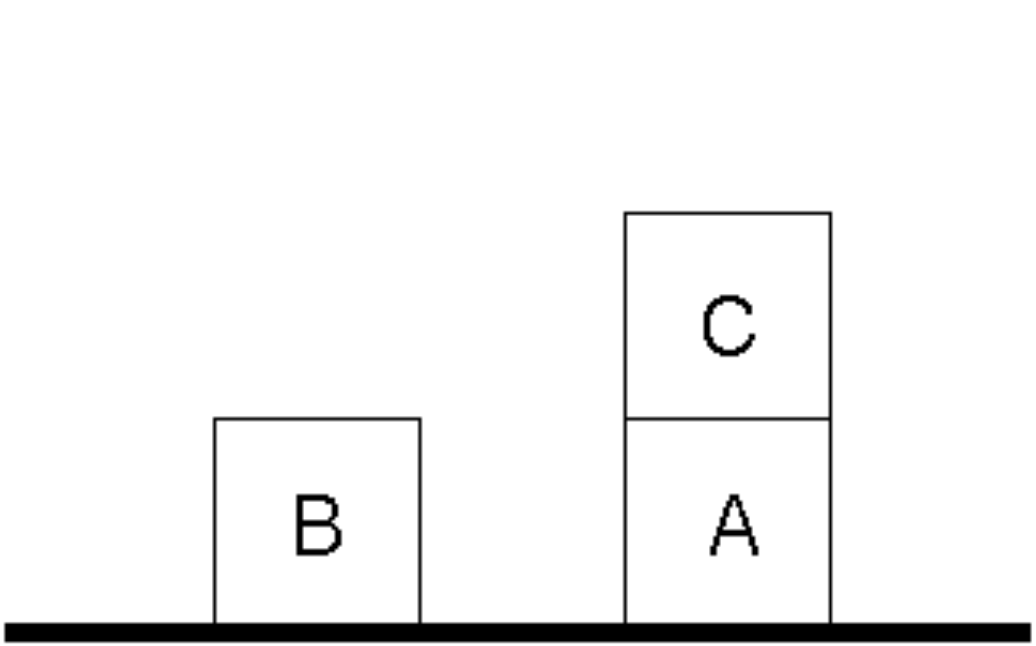
PRS beliefs

- the beliefs of a PRS agent consist of a set of state descriptions describing what is believed to be true at the current instant
- state description language is first order predicate calculus with the usual connectives $\wedge$, $\vee$ and $\neg$
- state descriptions can contain variables (assumed to be universally quantified)
- e.g., in a 'blocks world' domain state description (on blockB blockA) could represent the belief that blockA is on top of blockB

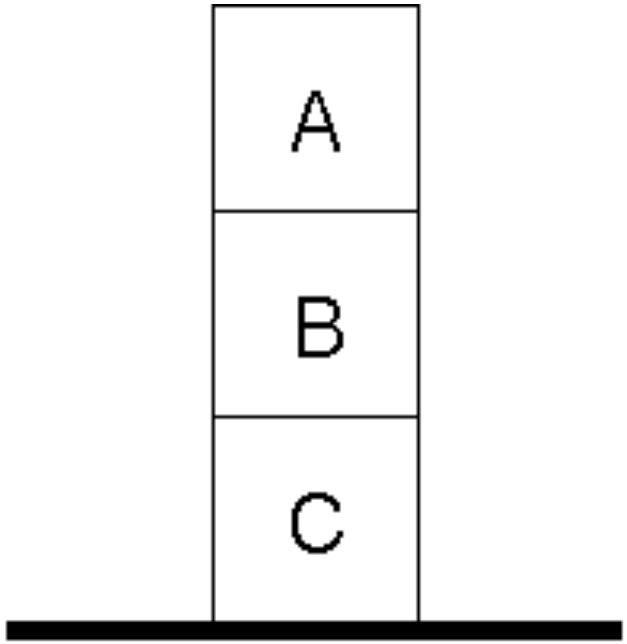
PRS goals

- goals are specified as a combination of a goal operator applied to a state description (possibly containing existentially quantified variables):
  - $(!p)$ where p is a state description is true of a sequence of states if p is true of the last state of the sequence, i.e, achievement goals
  - $(?p)$ is true of a sequence of states if p is true of the first state in the sequence, i.e., test goals
  - $(\#p)$ is true if p is preserved throughout the sequence of states (corresponding to the execution of a plan), i.e., maintenance goals
- behaviour descriptions can be combined using $\wedge$ and $\vee$, e.g., $(\#p \wedge !q)$ means maintain p until q becomes true

Blocks world



Start



Goal

Example: blocks world goals

- (! (*on blockA blockB*)) expresses an achievement goal of stacking *blockA* on *blockB*
- ($\wedge$ (! (*on blockB blockC*))(!(*onblockA blockB*))) expresses an achievement goal of building a tower of blocks
- (? (*clear blockA*)) expresses a test goal which tests whether there is a block on *blockA*
- (# (*clear blockA*)) expresses a maintenance goal of keeping *blockA* clear
- ($\wedge$ (# (*clear blockA*))(! (*holding blockA*))) expresses the goal of keeping *blockA* clear until it has been picked up

PRS plans

- a plan consists of an invocation condition and a body
- the *invocation condition* specifies the situations in which the plan is useful
- the plan *body* specifies the steps of the plan
- together the invocation condition and body express a declarative fact about the results and utility of performing a sequence of actions in a given situation

Plan invocation

- the invocation condition consists of a trigger and a context
- the *trigger* is a logical expression describing the events that must occur for the plan to be applicable, e.g.,
  - change in system goals (goal-directed invocation)
  - change in the system beliefs (data-directed or reactive invocation)
- the *context* is a logical expression specifying the conditions that must be true of the current state for the plan to be applicable

Plan body

- the body of a plan is a *graph* in which each edge is labelled with a (sub)goal to be achieved
- each step in a plan therefore involves the achievement of a subgoal
- allows the selection of the most appropriate means of achieving the subgoal in the current situation
- some plans have no bodies — such primitive plans are associated with an *action* (implemented in Lisp) that is directly performable by the system
- the execution of any plan ultimately reduces to the execution of a sequence of primitive plans

Plan execution

- execution begins at the start node, and proceeds by following edges through the graph until a finish node (a node with no outgoing edges) is reached
- to traverse an edge, either the associated goal must already have been achieved, or a plan must be executed which achieves the goal
- execution is nondeterministic — if PRS fails to traverse an edge from some node, other edges from that node (if any) may be tried
- if PRS fails to traverse any of the edges from a node, the plan fails
- a plan achieves a goal if its execution results in a behaviour that satisfies the goal description

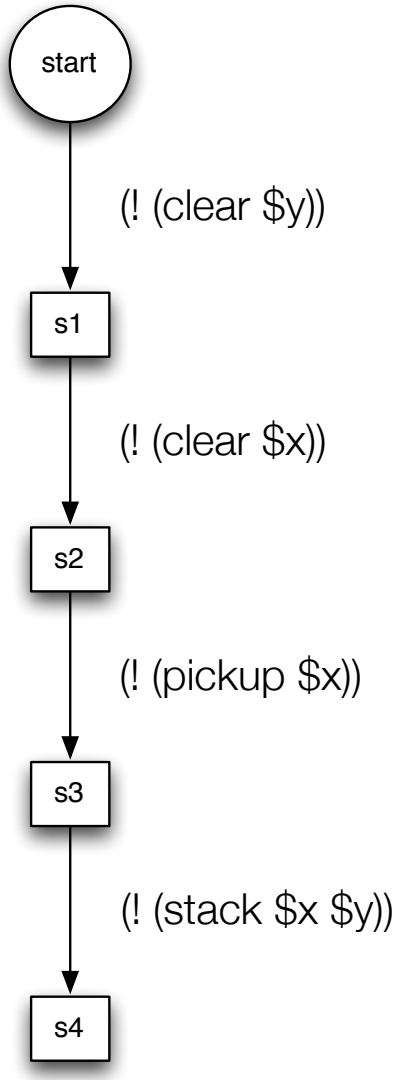
Example blocks world plan

trigger:

(!(*on* $x $y))

context:

(¬(*holding* $z))



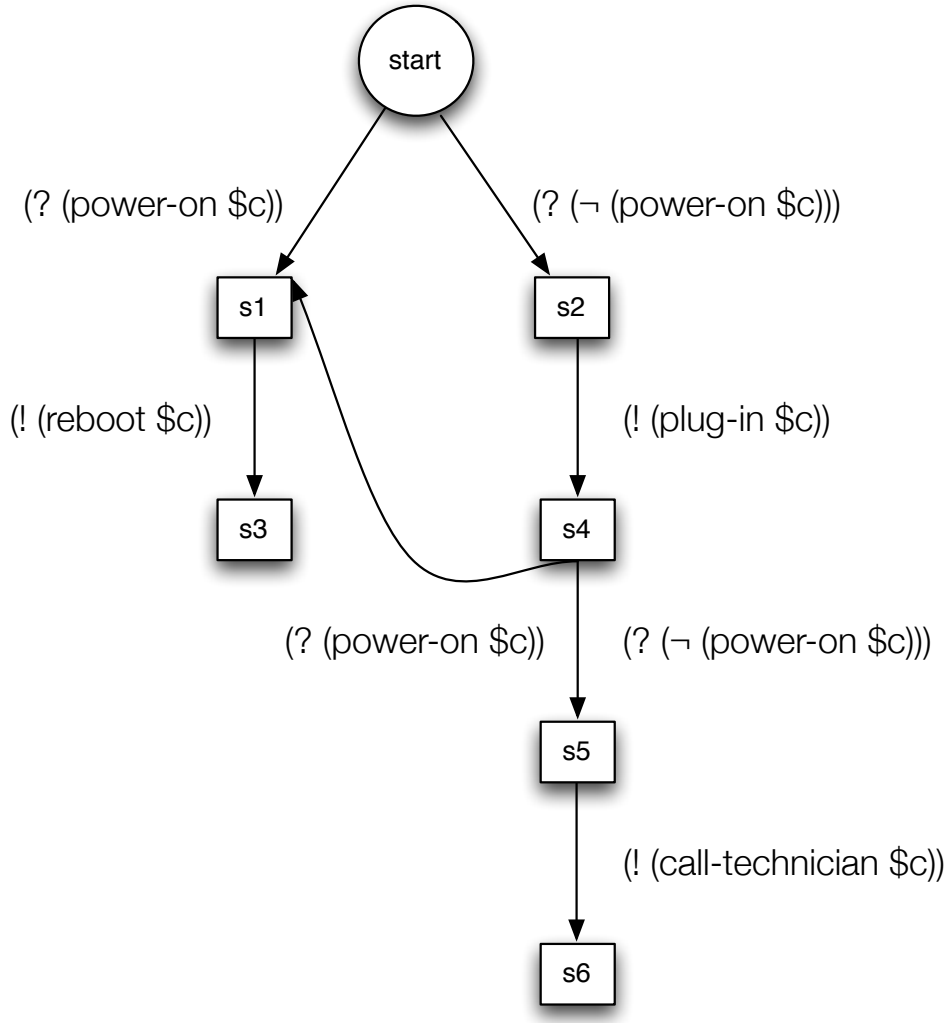
Example computer repair plan

trigger:

(!(*working-computer* $c))

context:

(?(¬(*burning-smell* $c)))



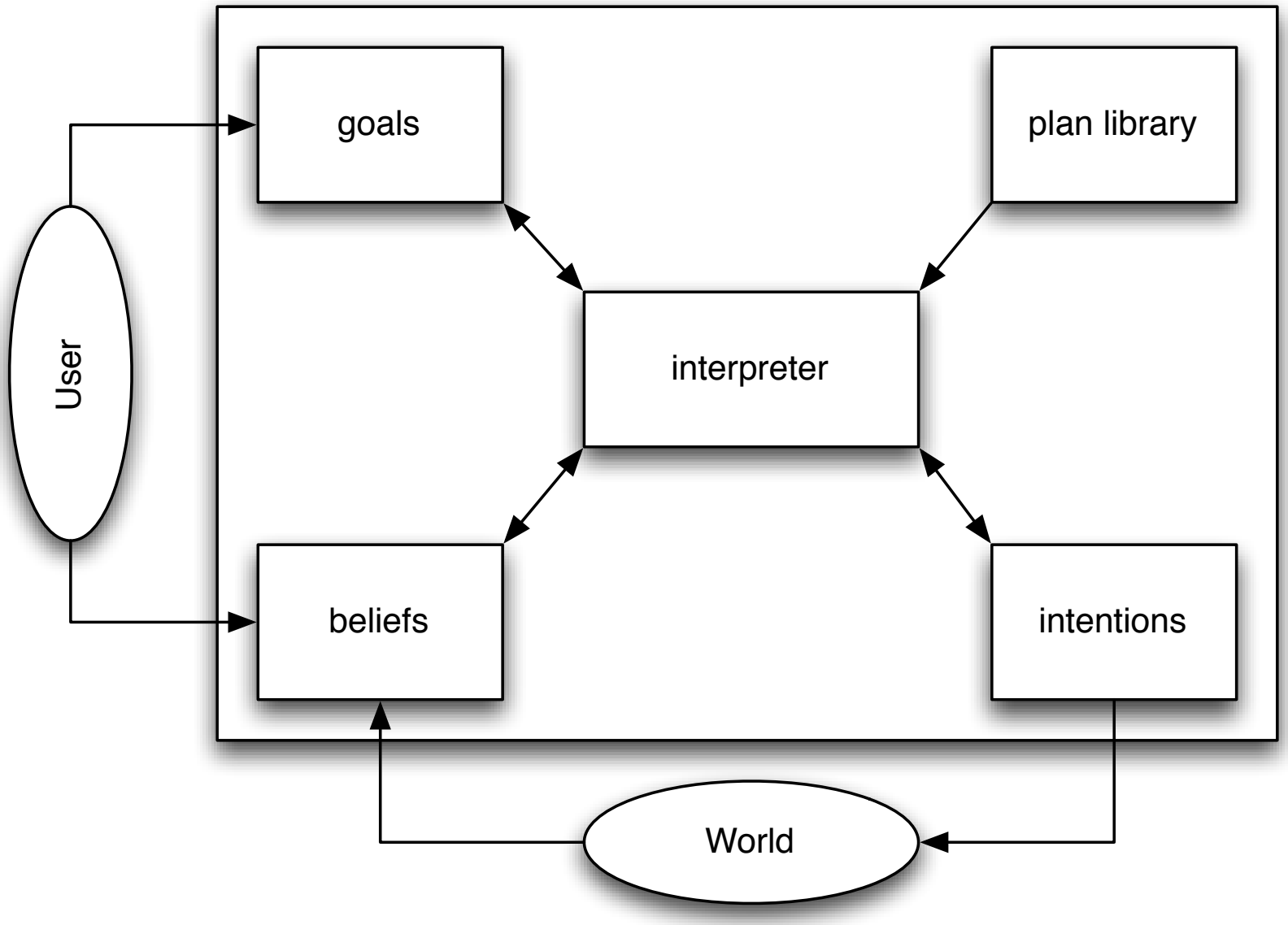
PRS intentions

- the intention structure contains all tasks chosen for execution, either now or at some future time
- each intention consists of a single top-level plan (invoked by a top-level goal or belief) together with all the sub-plans that have been invoked to achieve sub-goals of a parent plan
- intentions may be *active* (executable) or *suspended* (waiting for some condition to become true)
- e.g., an intention may suspend while a move action is being executed
- allows PRS to remain responsive while performing time consuming actions or deliberating

PRS interpreter

- the program of a PRS agent is executed by the PRS *interpreter*
- the interpreter operates in cycles — at each cycle new beliefs (from sensors or concluded by a plan) and goals (top-level goals from users or subgoals generated by a plan) can trigger plans
- the context of triggered plans is matched (unified) against the current beliefs and goals to determine which plans are applicable
- one or more applicable plans is chosen for execution by placing it on the intention structure
- interpreter then chooses an active intention and executes the next step in the associated plan

PRS architecture



PRS deliberation

- the default PRS interpreter cycle can be extended using metalevel plans
- *metalevel* plans encode methods for, e.g., choosing among multiple, applicable plans, determining how to achieve a conjunction or disjunction of goals, deciding which intention to execute next etc.
- metalevel plans are triggered by changes in the system's metalevel beliefs about system's current beliefs, goals, intentions and applicable plans
- applicable metalevel plans give rise to further metalevel beliefs and reflection continues until a single plan can be chosen and intended
- e.g., intention structure may contain a suspended object level plan which has posted a subgoal and a metalevel plan to decide how to accomplish the subgoal

Proactivity and reactivity

- unless a new goal or belief activates a new plan, PRS will attempt to execute any intentions it has already decided upon — i.e, it *commits* to its intentions in Bratman's sense
- however new beliefs (resulting from plan execution or changes in the environment) and goals (resulting from user requests) will cause PRS to reassess its current intentions, and perhaps choose to work on something else
- PRS can even modify its own reasoning process — e.g., it may decide that, given the current situation, it has no time for further deliberation and must act immediately

Reasoning in PRS

- the design of PRS represents a trade-off between *expressiveness* and *tractability*
- reasoning in PRS means the selection and execution of appropriate plans to achieve the agent's goals, given its current beliefs
- no built-in support for logical inference (though this can be programmed using plans)
- a key objective was support for the development of *real-time systems* — all the basic steps of the interpreter are guaranteed to terminate in finite time

The legacy of PRS

1990: AGENT-0 (Shoham)

1993: PLACA (Thomas; AGENT-0 extension with plans)

1996: PRS-Lite (Meyers)

1996: AgentSpeak(L) (Rao; inspired by PRS)

1996: Golog (Reiter, Levesque, Lesperance)

1997: 3APL (Hindriks et al.)

1998: ConGolog (De Giacomo, Levesque, Lesperance)

2000: JACK (Busetta, Howden, Ronnquist, Hodgson)

2000: GOAL (Hindriks et al.)

2002: Jason (Bordini, Hubner; implementation of AgentSpeak)

2004: SPARK (Morley, Meyers; PRS-Lite revisited)

2008: 2APL (successor to 3APL)

Key language features

AGENT-0:	Speech acts
AgentSpeak(L):	Events & intentions
Golog:	Action theories
3APL:	Rules for adopting, revising and dropping goals & plans
JACK:	Java integration
Jason:	AgentSpeak + communication, belief actions, atomic plans
2APL:	Events, modules, ...

Summary

- BDI agents are programmed in terms of beliefs, goals, plans and intentions
- agent *interpreter* or deliberation cycle enforces relationships between beliefs, goals and intentions and causes the agent to choose plans to achieve its goals based on its beliefs
- BDI languages are typically very high-level: e.g., logic-based, with well-defined operational semantics
- however most agent programming languages depart to some degree from the BDI model of agency defined by Bratman et al.

The next lecture

Programming multi-agent systems